Bridges

```
struct Edge {
    int x, y, num;

    Edge(int x1, int y1, int num1): x(x1), y(y1), num(num1) {}
};

bool operator<(Edge edge, Edge edge1) {
    return edge.num < edge1.num;
}

vector<vector<Edge>> d;
vector<int> tin, up;
vector<bool> was;
int time1 = 0;
set<Edge> bridge;

void bridges(int vertex, int num) {
    tin[vertex] = up[vertex] = time1++, was[vertex] = true;
    for (Edge q : d[vertex]) {
        if (!was[q.y]) {
            bridges(q.y, q.num);
            up[vertex] = min(up[vertex], up[q.y]);
            if (up[q.y] > tin[vertex]) {
                bridge.insert(q);
            }
        } else if (q.num != num) {
            up[vertex] = min(up[vertex], tin[q.y]);
        }
    }
}

vector<vector<pair<int, p>>> d1;
vector<vector<int>> comp;
vector<p> num;

void cond(int vertex, int now) {
    was[vertex] = false, num[vertex] = {now, comp[now].size()};
    comp[now].push_back(vertex);
    for (Edge q: d[vertex]) {
        if (!was[q.y] && bridge.find(q) != bridge.end()) {
            cond(q.y, now);
        } else if (!was[q.y]) {
            d1.emplace_back();
            d1[now].push_back({comp.size(), {q.x, q.y}});
            d1[comp.size()].push_back({now, {q.y, q.x}});
            comp.emplace_back();
            cond(q.y, (int)comp.size()-1);
        }
    }
}
```

CSS

```
vector<vector<int>> d, d1, cond, comp;
vector<int> who, topsort;
vector<bool> was;

void top_sort(int vertex) {
    was[vertex] = true;
    for (int q : d[vertex]) {
        if (!was[q]) {
            top_sort(q);
        }
    }
    topsort.push_back(vertex);
}

void DFS_CSS(int vertex) {
    was[vertex] = true, who[vertex] = (int)comp.size()-1;
    comp.back().push_back(vertex);
    for (int q : d1[vertex]) {
        if (!was[q]) {
            DFS_CSS(q);
        } else if (who[q] != who[vertex]) {
            cond[who[q]].push_back(who[vertex]);
        }
    }
}

void CSS() {
    int n = d.size();
    d1.assign(n, {}), comp = {}, cond = {}, who.assign(n, -1), topsort = {}, was.assign(n, false);
    for (int q = 0; q < n; q++) {
        if (!was[q]) {
            top_sort(q);
        }
        for (int q1 : d[q]) {
            d1[q1].push_back(q);
        }
    }
    reverse(topsort.begin(), topsort.end());
    was.assign(n, false);
    for (int q : topsort) {
        if (!was[q]) {
            comp.emplace_back(), cond.emplace_back();
            DFS_CSS(q);
        }
    }
}
```

Dinitz

```
struct Edge {
    int x, y, c, f;
};

struct Flow {
    vector<vector<int>> gf;
    vector<Edge> edges;

    explicit Flow(int n) {
        gf.assign(n, {});
    }

    void add_edge(int x, int y, int c, bool directed) {
        gf[x].push_back((int)edges.size());
        edges.emplace_back(x, y, c, 0);
        gf[y].push_back((int)edges.size());
        edges.emplace_back(y, x, (1-directed)*c, 0);
    }

    vector<int> layer, ind;

    bool build_layers(int x, int y) {
        layer.assign(gf.size(), -1);
        ind.assign(gf.size(), 0);
```

```
            queue<int> a;
            a.push(x);
            layer[x] = 0;
            while (!a.empty()) {
                int q = a.front();
                if (q == y) {
                    return true;
                }
                a.pop();
                for (int q1_ : gf[q]) {
                    Edge& q1 = edges[q1_];
                    if (q1.c == q1.f) {
                        continue;
                    }
                    if (layer[q1.y] == -1) {
                        a.push(q1.y);
                        layer[q1.y] = layer[q1.x]+1;
                    }
                }
            }
            return layer[y] != -1;
        }

        int push(int x, int y, int min1) {
            if (x == y) {
                return min1;
            }
            int ans = 0;
            for (; ind[x] < gf[x].size(); ind[x]++) {
                int num = gf[x][ind[x]];
                Edge& q = edges[num];
                if (layer[q.y] != layer[q.x]+1 || q.f == q.c) {
                    continue;
                }
                int pushed = push(q.y, y, min(min1, q.c-q.f));
                edges[num].f += pushed;
                edges[num ^ 1].f -= pushed;
                ans += pushed, min1 -= pushed;
                if (min1 == 0) {
                    return ans;
                }
            }
            return ans;
        }

        void build_flow(int x, int y) {
            while (build_layers(x, y)) {
                push(x, y, INF);
            }
        }

        int max_flow(int x, int y) {
            build_flow(x, y);
            int ans = 0;
            for (int q : gf[x]) {
                ans += edges[q].f;
            }
            return ans;
        }
    };
```

Min_cost

```
    struct Edge {
```

```
    int x, y, c, f, v;
};

struct Flow {
    vector<vector<int>> gf;
    vector<Edge> edges;

    explicit Flow(int n) {
        gf.assign(n, {});
    }

    void add_edge(int x, int y, int c, int v) {
        gf[x].push_back((int)edges.size());
        edges.emplace_back(x, y, c, 0, v);
        gf[y].push_back((int)edges.size());
        edges.emplace_back(y, x, 0, 0, -v);
    }

    vector<int> dists;

    void build_dists(int x) {
        int n = (int)gf.size();
        dists.assign(n, INF);
        vector<bool> taken(n, false);
        queue<int> a;
        dists[x] = 0, taken[x] = true;
        a.push(x);
        while (!a.empty()) {
            int q = a.front();
            a.pop();
            taken[q] = false;
            for (int q1 : gf[q]) {
                Edge& e = edges[q1];
                if (e.f != e.c && dists[e.y] > dists[q]+e.v) {
                    dists[e.y] = dists[q]+e.v;
                    if (!taken[e.y]) {
                        taken[e.y] = true;
                        a.push(e.y);
                    }
                }
            }
        }
    }

    bool push(int x, int y) {
        int n = (int)gf.size();
        vector<int> will(n, INF), parents(n, -1);
        priority_queue<p> a;
        will[x] = 0;
        a.emplace(0, x);
        while (!a.empty()) {
            int len = -a.top().first, q = a.top().second;
            a.pop();
            if (len != will[q]) {
                continue;
            }
            for (int q1 : gf[q]) {
                Edge& e = edges[q1];
                int will_dist = len+e.v+dists[e.x]-dists[e.y];
                if (e.f != e.c && will[e.y] > will_dist) {
                    will[e.y] = will_dist, parents[e.y] = q1;
                    a.emplace(-will_dist, e.y);
                }
            }
            }
            if (will[y] == INF) {
                return false;
            }
            while (x != y) {
                edges[parents[y]].f++;
                edges[parents[y] ^ 1].f--;
                y = edges[parents[y]].x;
            }
            for (int q = 0; q < n; q++) {
                will[q] -= dists[x]-dists[q];
            }
            dists = will;
            return true;
        }

        void build_flow(int x, int y, int k) {
            build_dists(x);
            for (int q = 0; q < k && push(x, y); q++);
        }

        int min_cost(int x, int y, int k = INF) {
            build_flow(x, y, k);
            int ans = 0;
            for (int q = 0; q < edges.size(); q += 2) {
                ans += edges[q].f*edges[q].v;
            }
            return ans;
        }

        vector<vector<int>> ways;

        void find_way(int x, int y) {
            if (x == y) {
                return;
            }
            for (int q1 : gf[x]) {
                Edge& q = edges[q1];
                if (q.f > 0) {
                    find_way(q.y, y);
                    edges[q1].f--, edges[q1 ^ 1].f++;
                    ways.back().push_back(q1);
                    return;
                }
            }
        }

        void decompose(int x, int y) {
            ways = {};
            vector<Edge> was_edges = edges;
            int k = 0;
            for (int q : gf[x]) {
                k += edges[q].f;
            }
            for (int q = 0; q < k; q++) {
                ways.emplace_back();
                find_way(x, y);
                reverse(ways.back().begin(), ways.back().end());
            }
            edges = was_edges;
        }
    };
```

Kun

```
vector<vector<int>> d;
vector<int> pa, pb;
vector<bool> was_a, was_b;

bool find_chain(int vertex) {
    was_a[vertex] = true;
    for (int q : d[vertex]) {
        if (pb[q] == -1 || !was_a[pb[q]] && find_chain(pb[q])) {
            pa[vertex] = q, pb[q] = vertex;
            return true;
        }
    }
    return false;
}

void Kun(int n, int m) {
    pa.assign(n, -1), pb.assign(m, -1), was_a.assign(n, false);
    for (int q = 0; q < n; q++) {
        if (find_chain(q)) {
            was_a.assign(n, false);
        }
    }
}

int max_matching(int n, int m) {
    Kun(n, m);
    return n-count(pa.begin(), pa.end(), -1);
}

void DFS_L_minus(int vertex) {
    was_a[vertex] = true;
    for (int q : d[vertex]) {
        if (q == pa[vertex] || was_b[q]) {
            continue;
        }
        was_b[q] = true;
        if (pb[q] != -1 && !was_a[pb[q]]) {
            DFS_L_minus(pb[q]);
        }
    }
}

int independent_set(int n, int m) {
    Kun(n, m);
    was_a.assign(n, false), was_b.assign(m, false);
    for (int q = 0; q < n; q++) {
        if (pa[q] == -1) {
            DFS_L_minus(q);
        }
    }
    return count(was_a.begin(), was_a.end(), true)+count(was_b.begin(), was_b.end(), false);
}

int paths_splitting(int n) {
    return n-max_matching(n, n);
}

void DFS_reachable(int vertex) {
    was_a[vertex] = true;
    for (int q : d[vertex]) {
        if (!was_a[q]) {
            DFS_reachable(q);
        }
    }
}

void make_transitive_closure(int n) {
    for (int q = 0; q < n; q++) {
        was_a.assign(n, false);
        DFS_reachable(q);
        d[q] = {};
        for (int q1 = 0; q1 < n; q1++) {
            if (q != q1 && was_a[q1]) {
                d[q].push_back(q1);
            }
        }
    }
}

int max_antichain(int n) {
    make_transitive_closure(n);
    return independent_set(n, n)-n; // was_a && !was_b
}
```

Centroid

```
vector<vector<int>> d;
vector<int> height, sizes, f;
vector<bool> was_centroid;

void make_size(int vertex, int parent, int h) {
    sizes[vertex] = 1, height[vertex] = h;
    f.push_back(vertex);
    for (int q : d[vertex]) {
        if (!was_centroid[q] && q != parent) {
            make_size(q, vertex, h+1);
            sizes[vertex] += sizes[q];
        }
    }
}

int centroid(int vertex) {
    int parent = -1, all = sizes[vertex];
    bool flag = false;
    while (!flag) {
        flag = true;
        for (int q : d[vertex]) {
            if (!was_centroid[q] && q != parent && sizes[q]*2 > all) {
                parent = vertex, vertex = q, flag = false;
                break;
            }
        }
    }
    return vertex;
}

void centroid_decomposition(int vertex) {
    make_size(vertex, -1, 0);
    int k = centroid(vertex);
    f = {};
    make_size(k, -1, 0);
    was_centroid[k] = true;
    for (int q : d[k]) {
        if (!was_centroid[q]) {
            centroid_decomposition(q);
        }
    }
}
```

HLD

```
struct DO {
    vector<int> do_arr, mod;
    int len;

    DO(vector<int> &a) {
        len = 1;
        while (len < a.size()) {
            len *= 2;
        }
        do_arr.assign(2*len, 0), mod.assign(2*len, 0);
        for (int q = len; q < len+a.size(); q++) {
            do_arr[q] = a[q-len];
        }
        for (int q = len-1; q > 0; q--) {
            do_arr[q] = do_arr[2*q]+do_arr[2*q+1];
        }
    }

    void push(int q, int length) {
        do_arr[2*q] += length/2*mod[q], do_arr[2*q+1] += length/2*mod[q];
        mod[2*q] += mod[q], mod[2*q+1] += mod[q], mod[q] = 0;
    }

    void update(int l, int r, int l1, int r1, int q, int x) {
        if (l1 >= r || l >= r1) {
            return;
        }
        if (l1 <= l && r <= r1) {
            do_arr[q] += x*(r-l), mod[q] += x;
            return;
        }
        push(q, r-l);
        int m = (l+r)/2;
        update(l, m, l1, r1, 2*q, x);
        update(m, r, l1, r1, 2*q+1, x);
        do_arr[q] = do_arr[2*q]+do_arr[2*q+1];
    }

    int ans(int l, int r, int l1, int r1, int q) {
        if (l1 >= r || l >= r1) {
            return 0;
        }
        if (l1 <= l && r <= r1) {
            return do_arr[q];
        }
        push(q, r-l);
        int m = (l+r)/2;
        return ans(l, m, l1, r1, 2*q)+ans(m, r, l1, r1, 2*q+1);
    }
};

vector<vector<int>> d;
vector<int> parent, sizes, tin, tout, height, order, up;
vector<int> a;

void for_HLD(int vertex, int p1) {
    int now = 1;
    for (int q : d[vertex]) {
        if (q != p1) {
                for_HLD(q, vertex);
                now += sizes[q];
            }
        }
        sizes[vertex] = now, parent[vertex] = p1;
        sort(d[vertex].begin(), d[vertex].end(), [](int x, int y) {return sizes[x] > sizes[y];});
    }

    void DFS_HLD(int vertex, int p1, int h) {
        tin[vertex] = order.size(), height[vertex] = h;
        up[vertex] = (p1 == -1 || d[p1][0] != vertex ? vertex : up[p1]);
        order.push_back(a[vertex]);
        for (int q : d[vertex]) {
            if (q != p1) {
                DFS_HLD(q, vertex, h+1);
            }
        }
        tout[vertex] = order.size();
    }

    DO make_HLD() {
        int n = d.size();
        sizes.assign(n, -1), parent.assign(n, -1), height.assign(n, -1);
        tin.assign(n, -1), tout.assign(n, -1), up.assign(n, -1);
        for_HLD(0, -1);
        DFS_HLD(0, -1, 0);
        return DO(order);
    }

    void update_way(int x, int y, int k, DO &do_arr) {
        if (height[up[x]] < height[up[y]]) {
            swap(x, y);
        }
        while (up[x] != up[y]) {
            do_arr.update(0, do_arr.len, tin[up[x]], tin[x]+1, 1, k);
            x = parent[up[x]];
            if (height[up[x]] < height[up[y]]) {
                swap(x, y);
            }
        }
        do_arr.update(0, do_arr.len, min(tin[x], tin[y]), max(tin[x], tin[y])+1, 1, k);
    }

    int ans_way(int x, int y, DO &do_arr) {
        if (height[up[x]] < height[up[y]]) {
            swap(x, y);
        }
        int ans = 0;
        while (up[x] != up[y]) {
            ans += do_arr.ans(0, do_arr.len, tin[up[x]], tin[x]+1, 1);
            x = parent[up[x]];
            if (height[up[x]] < height[up[y]]) {
                swap(x, y);
            }
        }
        ans += do_arr.ans(0, do_arr.len, min(tin[x], tin[y]), max(tin[x], tin[y])+1, 1);
        return ans;
    }
```

LCA_linear_memory

```
    vector<vector<int>> d;
    vector<int> parent, height, jump;

```

```
void make_LCA(int vertex, int p1, int h) {
    parent[vertex] = p1, height[vertex] = h;
    if (p1 != -1 && height[jump[p1]]-height[p1] == height[jump[jump[p1]]]-height[jump[p1]]) {
        jump[vertex] = jump[jump[p1]];
    } else {
        jump[vertex] = (p1 == -1 ? vertex : p1);
    }
    for (int q : d[vertex]) {
        if (q != p1) {
            make_LCA(q, vertex, h+1);
        }
    }
}

int k_ancestor(int vertex, int k) {
    int h = height[vertex]-k;
    while (height[vertex] > h) {
        vertex = (height[jump[vertex]] >= h ? jump[vertex] : parent[vertex]);
    }
    return vertex;
}

int LCA(int x, int y) {
    if (height[x] < height[y]) {
        swap(x, y);
    }
    x = k_ancestor(x, height[x]-height[y]);
    while (x != y) {
        if (jump[x] != jump[y]) {
            x = jump[x], y = jump[y];
        } else {
            x = parent[x], y = parent[y];
        }
    }
    return x;
}
```

Berlekamp

```
const int C = 1791179179;

int pow1(int x, int y) {
    if (y == 0) {
        return 1;
    }
    if (y % 2 == 0) {
        return pow1(x*x % C, y/2);
    }
    return pow1(x, y-1)*x % C;
}

vector<int> Berlekamp(vector<int> &rec) {
    int n = rec.size(), q1 = 0;
    while (q1 < n && rec[q1] == 0) {
        q1++;
    }
    if (q1 == n) {
        return {};
    }
    int t = rec[q1] % C, q2 = q1++;
    vector<int> was, now = vector<int>(q1, 0);
    for (; q1 < n; q1++) {
        int d = -rec[q1] % C;
        for (int q = 1; q <= now.size(); q++) {
            d = (d+now[q-1]*rec[q1-q]) % C;
        }
        if (d == 0) {
            continue;
        }
        vector<int> will = now;
        while (will.size() < q1-q2+(int)was.size()) {
            will.push_back(0);
        }
        int mul = d*pow1(t, C-2) % C;
        will[q1-q2-1] = (will[q1-q2-1]+mul) % C;
        for (int q = 0; q < was.size(); q++) {
            will[q1-q2+q] = (will[q1-q2+q]-was[q]*mul) % C;
        }
        was = now, now = will, t = d, q2 = q1;
    }
    for (int& q : now) {
        q = (q+C) % C, q -= (q > C/2)*C;
    }
    while (!now.empty() && now.back() == 0) {
        now.pop_back();
    }
    return now;
}

int stupid(int n) {
    //to do
}

int find_n(int n, bool flag = false) {
    int k = 57;
    vector<int> a = {0};
    for (int q = 1; q < (flag ? n+1 : k); q++) {
        a.push_back(stupid(q));
    }
    vector<int> rec = Berlekamp(a);
    if (flag) {
        for (int q : rec) {
            cout << q << ' ';
        }
        cout << endl;
    }
    for (int q = k; q <= n; q++) {
        a.push_back(0);
        for (int q1 = 1; q1 <= rec.size(); q1++) {
            a.back() += a[q-q1]*rec[q1-1] % C;
        }
        a.back() = (a.back() % C+C) % C;
    }
    return a[n];
}
```

Pollard

```
__int128 pow2(__int128 x, __int128 y, __int128 C) {
    if (y == 0) {
        return 1;
    }
    if (y % 2 == 0) {
        return pow2(x*x % C, y/2, C);
    }
    return pow2(x, y-1, C)*x % C;
}

```

```
bool Miller_Rabin_test(int a, int n) {
    int d = n-1;
    while ((d & 1) ^ 1) {
        d >>= 1;
    }
    __int128 now = pow2(a, d, n);
    if (now == 1) {
        return true;
    }
    while (d < n-1) {
        if (now == n-1) {
            return true;
        }
        now = now*now % n, d <<= 1;
    }
    return false;
}

mt19937 randint(17957179);

bool Miller_Rabin(int n, int k = 20) {
    if (n == 1) {
        return false;
    }
    for (int q = 0; q < k; q++) {
        if (!Miller_Rabin_test(randint() % (n-1)+1, n)) {
            return false;
        }
    }
    return true;
}

int f_Pollard(__int128 x, int n) {
    return (x*x+1) % n;
}

vector<int> make_Pollard(int n) {
    if (Miller_Rabin(n)) {
        return {n};
    }
    int x = randint() % (n-1)+1;
    int y = f_Pollard(x, n);
    while (__gcd(n, abs(y-x)) == 1) {
        x = f_Pollard(x, n);
        y = f_Pollard(f_Pollard(y, n), n);
    }
    if (x == y) {
        return make_Pollard(n);
    }
    int d = __gcd(n, abs(y-x));
    vector<int> ans = make_Pollard(d);
    for (int q : make_Pollard(n/d)) {
        ans.push_back(q);
    }
    return ans;
}

vector<int> Pollard(int n) {
    vector<int> primes, small = {2, 3, 5, 7};
    for (int q : small) {
        if (n % q == 0) {
            primes.push_back(q);
            while (n % q == 0) {
                n /= q;
            }
        }
    }
    if (n == 1) {
        return primes;
    }
    set<int> was;
    for (int q : make_Pollard(n)) {
        if (was.find(q) == was.end()) {
            primes.push_back(q);
            was.insert(q);
        }
    }
    return primes;
}
```

FFT_divide

```
int pow1(int x, int y, int C) {
    if (y == 0) {
        return 1;
    }
    if (y % 2 == 0) {
        return pow1(x*x % C, y/2, C);
    }
    return pow1(x, y-1, C)*x % C;
}

#define ld double
const ld PI = acosl(-1);

int reverse_bits(int x, int len) {
    int y = 0;
    for (int q = 0; q < len; q++) {
        y |= (((x >> q) & 1) << (len-q-1));
    }
    return y;
}

void FFT(vector<complex<ld>> &a) {
    int n = a.size(), len = 1;
    while ((1 << len) < n) {
        len++;
    }
    for (int q = 0; q < n; q++) {
        int q1 = reverse_bits(q, len);
        if (q < q1) {
            swap(a[q], a[q1]);
        }
    }
    for (int q = 1; q < n; q <<= 1) {
        complex<ld> root(cosl(PI/q), sinl(PI/q));
        for (int q1 = 0; q1 < n; q1 += (q << 1)) {
            complex<ld> now = 1;
            for (int q2 = q1; q2 < q1+q; q2++) {
                complex<ld> x = a[q2], y = a[q2+q]*now;
                a[q2] = x+y, a[q2+q] = x-y;
                now *= root;
            }
        }
    }
}

void IFFT(vector<complex<ld>> &a) {
        int n = a.size();
        FFT(a);
        reverse(a.begin()+1, a.end());
        for (complex<ld> &q : a) {
            q /= n;
        }
    }

    vector<complex<ld>> to_FFT_form(const vector<int> &a, int deg) {
        vector<complex<ld>> A;
        for (int q : a) {
            A.push_back(q);
        }
        while (A.size() < deg) {
            A.push_back(0);
        }
        return A;
    }

    vector<int> from_FFT_form(const vector<complex<ld>> &A) {
        vector<int> a;
        for (const complex<ld> &q : A) {
            a.push_back(roundl(q.real()));
        }
        while (a.size() > 1 && a.back() == 0) {
            a.pop_back();
        }
        return a;
    }

    vector<int> multiply(const vector<int> &a, const vector<int> &b, int MOD) {
        int deg = 1;
        while (deg < a.size()+b.size()) {
            deg <<= 1;
        }
        vector<complex<ld>> A = to_FFT_form(a, deg);
        vector<complex<ld>> B = to_FFT_form(b, deg);
        FFT(A), FFT(B);
        vector<complex<ld>> C(deg);
        for (int q = 0; q < deg; q++) {
            C[q] = A[q]*B[q];
        }
        IFFT(C);
        vector<int> c = from_FFT_form(C);
        for (int &q : c) {
            q %= MOD;
        }
        return c;
    }

    void fix_zeros(vector<int> &a, int n) {
        while (a.size() < n) {
            a.push_back(0);
        }
        while (a.size() > n && a.back() == 0) {
            a.pop_back();
        }
    }

    void modulo_x_n(vector<int> &a, int n) {
        while (a.size() < n) {
            a.push_back(0);
        }
        while (a.size() > n) {
            a.pop_back();
        }
    }

    int get_degree(int n) {
        int deg = 1;
        while (deg < n) {
            deg <<= 1;
        }
        return deg;
    }

    vector<int> opposite(vector<int> a, int MOD) {
        for (int &q : a) {
            q = (MOD-q) % MOD;
        }
        return a;
    }

    vector<int> find_reversed(vector<int> &a, int n, int MOD) {
        if (n == 1) {
            if (a[0] == 0) {
                exit(179);
            }
            return {pow1(a[0], MOD-2, MOD)};
        }
        vector<int> a0, a1;
        for (int q = 0; q < n/2; q++) {
            a0.push_back(a[q]);
        }
        for (int q = n/2; q < n; q++) {
            a1.push_back(a[q]);
        }
        vector<int> rev_a0 = find_reversed(a0, n/2, MOD);
        vector<int> Q = multiply(a0, rev_a0, MOD);
        fix_zeros(Q, n);
        reverse(Q.begin(), Q.end());
        modulo_x_n(Q, n/2);
        reverse(Q.begin(), Q.end());
        vector<int> rev_a0_by_a1 = multiply(rev_a0, a1, MOD);
        modulo_x_n(rev_a0_by_a1, n/2);
        for (int q = 0; q < n/2; q++) {
            Q[q] += rev_a0_by_a1[q];
        }
        vector<int> E = multiply(rev_a0, Q, MOD);
        modulo_x_n(E, n/2);
        E = opposite(E, MOD);
        for (int q : E) {
            rev_a0.push_back(q);
        }
        return rev_a0;
    }

    vector<int> reverse(vector<int> a, int n, int MOD) {
        int deg = get_degree(n);
        fix_zeros(a, deg);
        vector<int> rev = find_reversed(a, deg, MOD);
        modulo_x_n(rev, n);
        return rev;
    }

    vector<int> divide(vector<int> a, vector<int> b, int MOD) {
        if (a.size() < b.size() || a.size() == 1 && a[0] == 0) {
            return {0};
        }
        if (b.size() == 1 && b[0] == 0) {
            exit(179);
        }
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        vector<int> rev = reverse(b, (int)a.size()-(int)b.size()+1, MOD);
        vector<int> c = multiply(a, rev, MOD);
        modulo_x_n(c, (int)a.size()-(int)b.size()+1);
        reverse(c.begin(), c.end());
        return c;
    }

    vector<int> add(const vector<int> &a, const vector<int> &b, int MOD) {
        int n = a.size(), m = b.size();
        vector<int> c(max(n, m));
        for (int q = 0; q < max(n, m); q++) {
            c[q] = ((q < n ? a[q] : 0)+(q < m ? b[q] : 0)) % MOD;
        }
        fix_zeros(c, 1);
        return c;
    }

    vector<int> subtract(const vector<int> &a, const vector<int> &b, int MOD) {
        return add(a, opposite(b, MOD), MOD);
    }

    vector<int> take_remainder(const vector<int> &a, const vector<int> &b, int MOD) {
        vector<int> divider = divide(a, b, MOD);
        return subtract(a, multiply(b, divider, MOD), MOD);
    }

    vector<int> pow(vector<int> P, vector<int> mod, int n, int MOD) {
        if (n == 0) {
            return {1};
        }
        if (n % 2 == 0) {
            return pow(take_remainder(multiply(P, P, MOD), mod, MOD), mod, n/2, MOD);
        }
        return take_remainder(multiply(pow(P, mod, n-1, MOD), P, MOD), mod, MOD);
    }
```

FFT_modulo

```
    int pow1(int x, int y, int C) {
        if (y == 0) {
            return 1;
        }
        if (y % 2 == 0) {
            return pow1(x*x % C, y/2, C);
        }
        return pow1(x, y-1, C)*x % C;
    }

    vector<int> get_primes(int n) {
        vector<int> primes;
        int sqrt1 = sqrtl(n);
        for (int q = 2; q <= sqrt1; q++) {
            if (n % q == 0) {
                primes.push_back(q);
            }
            while (n % q == 0) {
                n /= q;
            }
        }
        if (n > 1) {
            primes.push_back(n);
        }
        return primes;
    }

    int find_g(int C) {
        vector<int> primes = get_primes(C-1);
        for (int q = 1;; q++) {
            bool flag = true;
            for (int q1 : primes) {
                if (pow1(q, (C-1)/q1, C) == 1) {
                    flag = false;
                    break;
                }
            }
            if (flag) {
                return q;
            }
        }
    }

    int reverse_bits(int x, int len) {
        int y = 0;
        for (int q = 0; q < len; q++) {
            y |= (((x >> q) & 1) << (len-q-1));
        }
        return y;
    }

    void FFT(vector<int> &a, int C) {
        int n = a.size(), len = 1;
        while ((1 << len) < n) {
            len++;
        }
        for (int q = 0; q < n; q++) {
            int q1 = reverse_bits(q, len);
            if (q < q1) {
                swap(a[q], a[q1]);
            }
        }
        int g = find_g(C);
        for (int q = 1; q < n; q <<= 1) {
            int root = pow1(g, (C-1)/(q << 1), C);
            for (int q1 = 0; q1 < n; q1 += (q << 1)) {
                int now = 1;
                for (int q2 = q1; q2 < q1+q; q2++) {
                    int x = a[q2], y = a[q2+q]*now % C;
                    a[q2] = (x+y) % C, a[q2+q] = (x-y+C) % C;
                    now = now*root % C;
                }
            }
        }
    }

    void IFFT(vector<int> &a, int C) {
        int n = a.size();
        FFT(a, C);
        reverse(a.begin()+1, a.end());
        for (int &q : a) {
            q = q*pow1(n, C-2, C) % C;
        }
    }
```

```
void fix_polynomial(vector<int> &a, int n, int C) {
    while (a.size() < n) {
        a.push_back(0);
    }
    while (a.size() > n && a.back() == 0) {
        a.pop_back();
    }
    for (int &q : a) {
        q = (q % C+C) % C;
    }
}

int get_degree(int n) {
    int deg = 1;
    while (deg < n) {
        deg <<= 1;
    }
    return deg;
}

vector<int> multiply(vector<int> a, vector<int> b, int C) {
    int deg = get_degree(a.size()+b.size());
    fix_polynomial(a, deg, C), fix_polynomial(b, deg, C);
    FFT(a, C), FFT(b, C);
    for (int q = 0; q < deg; q++) {
        a[q] = a[q]*b[q] % C;
    }
    IFFT(a, C);
    fix_polynomial(a, 1, C);
    return a;
}
```

C_k_n_modulo_p

```
int pow1(int x, int y, int C) {
    if (y == 0) {
        return 1;
    }
    if (y % 2 == 0) {
        return pow1(x*x % C, y/2, C);
    }
    return pow1(x, y-1, C)*x % C;
}

vector<int> fact, rev_fact;

void make_fact(int C) {
    fact = {1};
    for (int q = 1; q < C; q++) {
        fact.push_back(fact.back()*q % C);
    }
    rev_fact = {pow1(fact.back(), C-2, C)};
    for (int q = C-1; q > 0; q--) {
        rev_fact.push_back(rev_fact.back()*q % C);
    }
    reverse(rev_fact.begin(), rev_fact.end());
}

int get_fact(int n, int C) {
    if (n < C) {
        return fact[n];
    }
    int ans = fact[n % C]*pow1(fact.back(), n/C, C) % C;
    return ans*get_fact(n/C, C) % C;
}

int get_rev_fact(int n, int C) {
    if (n < C) {
        return rev_fact[n];
    }
    int ans = rev_fact[n % C]*pow1(rev_fact.back(), n/C, C) % C;
    return ans*get_rev_fact(n/C, C) % C;
}

int p_degree(int n, int C) {
    int ans = 0, deg = C;
    while (deg <= n) {
        ans += n/deg, deg *= C;
    }
    return ans;
}

int C_k_n(int k, int n, int C) {
    if (k < 0 || k > n) {
        return 0;
    }
    if (p_degree(n, C) > p_degree(k, C)+p_degree(n-k, C)) {
        return 0;
    }
    return get_fact(n, C)*get_rev_fact(k, C) % C*get_rev_fact(n-k, C) % C;
}
```

N_th_root

```
int pow1(int x, int y, int C) {
    if (y == 0) {
        return 1;
    }
    if (y % 2 == 0) {
        return pow1(x*x % C, y/2, C);
    }
    return pow1(x, y-1, C)*x % C;
}

vector<int> factor(int n) {
    vector<int> primes;
    int sqrt1 = sqrt(n);
    for (int q = 2; q <= sqrt1; q++) {
        if (n % q == 0) {
            primes.push_back(q);
        }
        while (n % q == 0) {
            n /= q;
        }
    }
    if (n > 1) {
        primes.push_back(n);
    }
    return primes;
}

int find_g(int C) {
    vector<int> primes = factor(C-1);
    for (int q = 1;; q++) {
        bool flag = true;
        for (int q1 : primes) {
            if (pow1(q, (C-1)/q1, C) == 1) {
                flag = false;
                break;
            }
        }
        if (flag) {
            return q;
        }
    }
}

int g_degree(int x, int g, int C) {
    int sqrt1 = sqrt(C);
    unordered_map<int, int> a;
    a.reserve(sqrt1);
    int now = 1;
    for (int q = 0; q < sqrt1; q++) {
        a[now] = q;
        now = now*g % C;
    }
    int rev_sqrt = pow1(now, C-2, C);
    for (int q = 0; q <= C/sqrt1; q++) {
        if (a.find(x) != a.end()) {
            return q*sqrt1+a[x];
        }
        x = x*rev_sqrt % C;
    }
    return -1;
}

int phi(int n) {
    vector<int> fact = factor(n);
    int ans = n;
    for (int q : fact) {
        ans -= ans/q;
    }
    return ans;
}

p ax_by_c(int a, int b, int c) {
    int t = __gcd(a, b);
    if (c % t != 0) {
        return {-1, -1};
    }
    a /= t, b /= t, c /= t;
    int x = (c*pow1(a, phi(b)-1, b) % b+b) % b;
    int y = (c-a*x)/b;
    return {x, y};
}

int sqrt_b(int a, int b, int C) {
    if (a == 0) {
        return 0;
    }
    int g = find_g(C);
    int deg_a = g_degree(a, g, C);
    int x = ax_by_c(b, C-1, deg_a).first;
    return (x == -1 ? -1 : pow1(g, x, C));
}
```

double

```
#define ld long double
const ld E = 1e-8;

```

```
struct Pt_ld {
    ld x, y;

    Pt_ld(): x(0), y(0) {}
    Pt_ld(ld x, ld y): x(x), y(y) {}

    bool operator==(const Pt_ld& point) const {
        return abs(x-point.x) < E && abs(y-point.y) < E;
    }

    bool operator<(const Pt_ld& point) const {
        return x+E < point.x || abs(x-point.x) < E && y+E < point.y;
    }

    Pt_ld operator+(const Pt_ld& point) const {
        return {x+point.x, y+point.y};
    }

    Pt_ld operator-(const Pt_ld& point) const {
        return {x-point.x, y-point.y};
    }

    ld get_abs() const {
        return hypotl(x, y);
    }

    Pt_ld norm(ld len) const {
        ld k = len/get_abs();
        return {x*k, y*k};
    }
};

istream& operator>>(istream& in, Pt_ld& point) {
    in >> point.x >> point.y;
    return in;
}

ld dot(Pt_ld x, Pt_ld y) {
    return x.x*y.x+x.y*y.y;
}

ld cross(Pt_ld x, Pt_ld y) {
    return x.x*y.y-x.y*y.x;
}

ld abs(Pt_ld x) {
    return x.get_abs();
}

ld abs_2(Pt_ld x) {
    return x.x*x.x+x.y*x.y;
}

ld dist(Pt_ld x, Pt_ld y) {
    return abs(x-y);
}

ld dist_2(Pt_ld x, Pt_ld y) {
    return abs_2(x-y);
}

ld angle(Pt x, Pt y) {
    // angle between (1, 0) и (sin, cos)
    return atan2(cross(x, y), dot(x, y));
}

struct Line_ld {
    ld a, b, c;

    Line_ld(Pt_ld x, Pt_ld y): a(x.y-y.y), b(y.x-x.x), c(cross(x, y)) {}

    Pt_ld dir() const {
        return {-b, a};
    }

    Pt_ld norm() const {
        return {a, b};
    }

    Line_ld per(Pt_ld x) const {
        return {x, x+norm()};
    }

    bool on(Pt_ld x) const {
        return abs(a*x.x+b*x.y+c) < E;
    }
};

bool on_line(Pt_ld x, Pt_ld y, Pt_ld z) {
    return abs(cross(y-x, z-x)) < E;
}

bool is_parallel(Line_ld a, Line_ld b) {
    return abs(cross(a.dir(), b.dir())) < E;
}

vector<Pt_ld> inter(Line_ld a, Line_ld b) {
    if (is_parallel(a, b)) {
        return {};
    }
    ld det = cross(a.norm(), b.norm());
    ld det_x = cross(Pt_ld(-a.c, a.b), Pt_ld(-b.c, b.b));
    ld det_y = cross(Pt_ld(a.a, -a.c), Pt_ld(b.a, -b.c));
    return {Pt_ld(det_x/det, det_y/det)};
}

Pt_ld projection(Line_ld line, Pt_ld x) {
    return inter(line, line.per(x))[0];
}

struct Seg_ld {
    Pt_ld x, y;

    Seg_ld() = default;
    Seg_ld(Pt_ld x, Pt_ld y): x(x), y(y) {}

    explicit operator bool() const {
        return x != y;
    }

    Line_ld line() const {
        return {x, y};
    }

    bool on(Pt_ld point) const {
        return on_line(x, y, point) && dot(point-q.x, point-q.y) < E;
    }
};
```

```

vector<Pt_ld> inter(Seg_ld a, Seg_ld b) {
    vector<Pt_ld> inters;
    if (!a && b.on(a.x)) {
        inters.push_back(a.x);
    } else if (!b && a.on(b.x)) {
        inters.push_back(b.x);
    }
    if (!a || !b) {
        return inters;
    }
    inters = inter(a.line(), b.line());
    if (!inters.empty() && a.on(inters[0]) && b.on(inters[0])) {
        return inters;
    }
    return {};
}

struct Cir_ld {
    Pt_ld x;
    ld r = 0;

    Cir_ld() = default;
    Cir_ld(Pt_ld x, ld r): x(x), r(r) {}

    bool on(Pt_ld point) const {
        return abs(dist(point, x)-r) < E;
    }
};

vector<Pt_ld> inter(Cir_ld cir, Line_ld line) {
    Pt_ld proj = projection(line, cir.x);
    ld h = dist(proj, cir.x);
    if (h > cir.r-E) {
        return h > cir.r+E ? vector<Pt_ld>() : vector{proj};
    }
    ld len = sqrtl(cir.r*cir.r-h*h);
    Pt_ld dir = line.dir().norm(len);
    vector<Pt_ld> ans = {proj-dir, proj+dir};
    sort(ans.begin(), ans.end());
    return ans;
}

vector<Pt_ld> inter(Cir_ld cir, Seg_ld seg) {
    if (!seg) {
        return cir.on(seg.x) ? vector{seg.x} : vector<Pt_ld>();
    }
    vector<Pt_ld> ans, inters = inter(cir, seg.line());
    for (Pt_ld q : inters) {
        if (seg.on(q)) {
            ans.push_back(q);
        }
    }
    return ans;
}
```

polygons

```
#define ld long double

struct Pt {
    int x, y;

    Pt(): x(0), y(0) {}
```

```
        Pt(int x, int y): x(x), y(y) {}

        auto operator<=>(const Pt& other) const = default;

        Pt operator+(const Pt& point) const {
            return {x+point.x, y+point.y};
        }

        Pt operator-(const Pt& point) const {
            return {x-point.x, y-point.y};
        }
    };

    istream& operator>>(istream& in, Pt& point) {
        in >> point.x >> point.y;
        return in;
    }

    int dot(Pt x, Pt y) {
        return x.x*y.x+x.y*y.y;
    }

    int cross(Pt x, Pt y) {
        return x.x*y.y-x.y*y.x;
    }

    ld abs(Pt x) {
        return hypot(x.x, x.y);
    }

    int abs_2(Pt x) {
        return x.x*x.x+x.y*x.y;
    }

    ld dist(Pt x, Pt y) {
        return abs(x-y);
    }

    int dist_2(Pt x, Pt y) {
        return abs_2(x-y);
    }

    ld angle(Pt x, Pt y) {
        // angle between (1, 0) и (sin, cos)
        return atan2(cross(x, y), dot(x, y));
    }

    bool on_line(Pt x, Pt y, Pt z) {
        return cross(y-x, z-x) == 0;
    }

    struct Seg {
        Pt x, y;

        Seg(Pt x, Pt y): x(x), y(y) {}
    };

    bool on_seg(Seg q, Pt point) {
        return on_line(q.x, q.y, point) && dot(point-q.x, point-q.y) <= 0;
    }

    int S_tr_2(Pt x, Pt y, Pt z) {
        return cross(y-x, z-x);
    }

    struct Polygon {
        vector<Pt> a;

        explicit Polygon(vector<Pt> a_): a(std::move(a_)) {
            if (a.size() == 2 && a[0] == a[1]) {
                a.pop_back();
            }
            if (a.size() < 3) {
                if (a.size() == 2) {
                    a.push_back(a[0]);
                }
                return;
            }
            normalize();
            a.push_back(a[0]);
        }

        void normalize() {
            int n = (int)a.size();
            int ind = 1;
            while (ind < n-1 && on_line(a[0], a[ind], a[ind+1])) {
                ind++;
            }
            assert(ind != n-1);
            rotate(a.begin(), a.begin()+ind, a.end());
            a.push_back(a[0]);
            vector<Pt> will_a = {a[0]};
            for (int q = 1; q < n; q++) {
                if (!on_line(will_a.back(), a[q], a[q+1])) {
                    will_a.push_back(a[q]);
                }
            }
            a = will_a;
        }

        vector<Seg> get_edges() const {
            int n = (int)a.size();
            if (n <= 1) {
                return {};
            }
            vector<Seg> edges;
            for (int q = 1; q < n; q++) {
                edges.emplace_back(a[q-1], a[q]);
            }
            return edges;
        }

        bool on_border(Pt point) const {
            return ranges::any_of(get_edges(), [point](Seg q) {return on_seg(q, point);});
        }

        int belonging(Pt point) const {
            if (on_border(point)) {
                return 2;
            }
            ld ang = 0;
            for (Seg q : get_edges()) {
                ang += angle(q.x-point, q.y-point);
            }
            // will be 0 or 2*pi
            return abs(ang) > numbers::pi_v<ld>;
        }

        int S_2() const {
            int ans = 0;
            for (Seg q : get_edges()) {
                ans += S_tr_2(a[0], q.x, q.y);
            }
            return ans;
        }
    };

    bool need_pop_back(Pt x, vector<Pt>& ans, bool up) {
        int m = (int)ans.size(), sign = 2*up-1;
        return m >= 2 && cross(ans[m-1]-ans[m-2], x-ans[m-2])*sign >= 0;
    }

    void make_envelope(vector<Pt>& a, bool up) {
        int n = (int)a.size();
        vector<Pt> ans = {a[0]};
        for (int q = 1; q < n; q++) {
            while (need_pop_back(a[q], ans, up)) {
                ans.pop_back();
            }
            ans.push_back(a[q]);
        }
        a = ans;
    }

    Polygon convex_hull(vector<Pt> a) {
        int n = (int)a.size();
        sort(a.begin(), a.end());
        if (n == 0 || a[0] == a.back()) {
            return Polygon(n == 0 ? vector<Pt>{} : vector{a[0]});
        }
        vector<Pt> up = {a[0]}, down = {a[0]};
        for (int q = 1; q < n-1; q++) {
            int value = cross(a.back()-a[0], a[q]-a[0]);
            if (value > 0) {
                up.push_back(a[q]);
            } else if (value < 0) {
                down.push_back(a[q]);
            }
        }
        up.push_back(a.back());
        down.push_back(a.back());
        make_envelope(up, true);
        make_envelope(down, false);
        up.insert(up.end(), down.rbegin()+1, down.rend()-1);
        return Polygon(up);
    }
```

Binary_Gauss

```
    const int MAX_N = 301, MAX_M = 300;

    struct Gauss {
        vector<bitset<MAX_M>> a;
        vector<bitset<MAX_N>> s;
        vector<bool> was;
        int n, C;

        Gauss(vector<string> &a1) {
            n = a1.size(), C = a1[0].size();
            a.assign(n, 0), s.assign(n, 0), was.assign(n, false);
            for (int q = 0; q < n; q++) {
                s[q][q] = true;
            for (int q1 = 0; q1 < C; q1++) {
                a[q][q1] = (a1[q][q1] == ’1’);
            }
        }
    }

    void subtract(int q, int q1) {
        for (int q2 = 0; q2 < n; q2++) {
            if (q != q2 && a[q2][q1]) {
                a[q2] ^= a[q], s[q2] ^= s[q];
            }
        }
    }

    void gauss() {
        int cnt = 0, q1 = 0;
        while (cnt < n && q1 < C) {
            int ind = -1;
            for (int q2 = 0; q2 < n; q2++) {
                if (!was[q2] && a[q2][q1]) {
                    ind = q2;
                    break;
                }
            }
            if (ind == -1) {
                q1++;
                continue;
            }
            subtract(ind, q1++);
            was[ind] = true, cnt++;
        }
    }
};
```

SLAE_solve_module

```
const int C = 1000000007;

int pow1(int x, int y) {
    if (y == 0) {
        return 1;
    }
    if (y % 2 == 0) {
        return pow1(x*x % C, y/2);
    }
    return pow1(x, y-1)*x % C;
}

struct Matrix {
    vector<vector<int>> a;
    int n, m;

    Matrix(vector<vector<int>> &a1) {
        n = a1.size(), m = a1[0].size(), a = a1;
    }

    void subtract(int q, int q1) {
        int del = pow1(a[q][q1], C-2);
        vector<int> norm;
        for (int q3 = 0; q3 < m; q3++) {
            if (a[q][q3] != 0) {
                norm.push_back(q3);
            }
        }
|            for (int q2 = q+1; q2 < n; q2++) {
|                if (a[q2][q1] != 0) {
|                    int coef = a[q2][q1]*del % C;
|                    for (int q3 : norm) {
|                        a[q2][q3] -= a[q][q3]*coef % C;
|                        a[q2][q3] += (a[q2][q3] < 0)*C;
|                    }
|                }
|            }
|        }
|
|        void gauss() {
|            int q = 0, q1 = 0;
|            while (q < n && q1 < m) {
|                for (int q2 = q; q2 < n; q2++) {
|                    if (a[q2][q1] != 0) {
|                        swap(a[q], a[q2]);
|                        break;
|                    }
|                }
|                if (a[q][q1] == 0) {
|                    q1++;
|                    continue;
|                }
|                subtract(q++, q1++);
|            }
|        }
|
|        vector<int> solve() {
|            gauss();
|            int q = n;
|            bool flag = true;
|            while (flag && (q--) > 0) {
|                for (int q1 : a[q]) {
|                    flag &= (abs(q1) == 0);
|                }
|            }
|            if (q == -1) {
|                return {C};
|            }
|            flag = true;
|            for (int q1 = 0; q1 < m-1; q1++) {
|                flag &= (a[q][q1] == 0);
|            }
|            if (flag) {
|                return {};
|            }
|            if (q < n-1 || n != m-1) {
|                return {C};
|            }
|            vector<int> ans(n);
|            for (; q > -1; q--) {
|                int sum1 = a[q][m-1];
|                for (int q1 = q+1; q1 < m-1; q1++) {
|                    sum1 -= a[q][q1]*ans[q1] % C;
|                }
|                ans[q] = (sum1 % C+C) % C*pow1(a[q][q], C-2) % C;
|            }
|            return ans;
|        }
|    };
```

Or_And_convolution

```
template <bool Rev = false>
vector<int> SOS(vector<int> a) {
    int n = (int)a.size();
    for (int q1 = 1; q1 < n; q1 <<= 1) {
        for (int q2 = q1; q2 < n; q2 += (q1 << 1)) {
            for (int q = q2; q < q2+q1; q++) {
                if constexpr (!Rev) {
                    a[q] += a[q ^ q1];
                } else {
                    a[q] -= a[q ^ q1];
                }
            }
        }
    }
    for (int& q : a) {
        q = (q % C+C) % C;
    }
    return a;
}

vector<int> num_ones;

void make_num_ones(int n) {
    num_ones.assign(1 << n, 0);
    for (int q = 0; q < n; q++) {
        num_ones[1 << q] = 1;
    }
    for (int q = 1; q < (1 << n); q++) {
        int last_bit = q-(q & (q-1));
        num_ones[q] = num_ones[q ^ last_bit]+num_ones[last_bit];
    }
}

int SOS_value(int q, const vector<int> &a) {
    int ans = (1-((num_ones[q] & 1) << 1))*a[0];
    for (int q1 = q; q1 > 0; q1 = ((q1-1) & q)) {
        ans += (1-((num_ones[q ^ q1] & 1) << 1))*a[q1];
    }
    return (ans % C+C) % C;
}

auto or_convolution_SOS(const vector<int> &a, const vector<int> &b) {
    int n = (int)a.size();
    vector<int> c(n);
    for (int q = 0; q < n; q++) {
        c[q] = a[q]*b[q] % C;
    }
    return c;
}

vector<int> or_convolution(const vector<int>& a, const vector<int>& b) {
    return SOS<true>(or_convolution_SOS(SOS(a), SOS(b)));
}

vector<int> and_convolution(vector<int> a, vector<int> b) {
    int n = (int)a.size(), ALL = n-1;
    for (int q = 0; q < (n >> 1); q++) {
        swap(a[q], a[q ^ ALL]);
        swap(b[q], b[q ^ ALL]);
    }
    vector<int> c = or_convolution(a, b);
    for (int q = 0; q < (n >> 1); q++) {
        swap(c[q], c[q ^ ALL]);
    }
    return c;
}
```

Smiths_theory

```
vector<int> get_smith(vector<vector<int>> &d) {
    int n = (int)d.size();
    vector<vector<int>> d1(n);
    for (int q = 0; q < n; q++) {
        for (int q1 : d[q]) {
            d1[q1].push_back(q);
        }
    }
    vector<int> smith(n, -1), all(n);
    iota(all.begin(), all.end(), 0);
    bool continue1 = true;
    for (int nim = 0; continue1; nim++) {
        continue1 = false;
        vector<int> num(n, 0);
        for (int q : all) {
            for (int q1 : d[q]) {
                num[q] += (smith[q1] == -1);
            }
        }
        vector<p> is;
        vector<bool> is_move(n, false), is_now(n, false);
        for (int q : all) {
            is.emplace_back(num[q], q);
            is_now[q] = true;
        }
        sort(is.rbegin(), is.rend());
        vector<int> ind(n);
        for (int q = 0; q < is.size(); q++) {
            ind[is[q].second] = q;
        }
        all = {};
        while (!is.empty() && is.back().first == 0) {
            continue1 = true;
            int vertex = is.back().second;
            is.pop_back();
            if (!is_now[vertex]) {
                continue;
            }
            is_now[vertex] = false, smith[vertex] = nim;
            for (int q : d1[vertex]) {
                if (is_move[q] || smith[q] != -1) {
                    continue;
                }
                if (is_now[q]) {
                    all.push_back(q);
                    is_now[q] = false;
                }
                is_move[q] = true;
                for (int q1 : d1[q]) {
                    if (!is_now[q1]) {
                        continue;
                    }
                    auto w = lower_bound(is.rbegin(), is.rend(), p(num[q1], -INF));
                    auto w1 = is.rbegin()+(int)is.size()-ind[q1]-1;
                    w1->first--, num[q1]--;
                    swap(*w, *w1);
                    swap(ind[w->second], ind[w1->second]);
                }
            }
        }
    }
    return smith;
}

int sum_games_result(vector<vector<int>> &d1, vector<vector<int>> &d2,
                     vector<int> &smith1, vector<int> &smith2, int x, int y) {
    if (smith1[x] == -1 && smith2[y] == -1) {
        return -1;
    }
    if (smith1[x] != -1 && smith2[y] != -1) {
        return (smith1[x] ^ smith2[y]) == 0;
    }
    bool was_swap = false;
    if (smith1[x] != -1) {
        swap(d1, d2);
        swap(smith1, smith2);
        was_swap = true;
        swap(x, y);
    }
    bool flag = false;
    for (int q : d1[x]) {
        flag |= (smith1[q] == smith2[y]);
    }
    if (was_swap) {
        swap(d1, d2);
        swap(smith1, smith2);
    }
    return flag-1;
}
```

Annealing_simulation

```
mt19937 randint(179);

int change(int ind, vector<int> &a) {
    int x = a[ind], cost = 0;
    //code
    return cost;
}

const int MN = (1LL << 20);

bool P(int x, int y, long double t) {
    long double is = exp((y-x)/t);
    return randint() % MN < is*MN;
}

vector<int> annealing(int n) {
    vector<int> a(n);
    iota(a.begin(), a.end(), 0);
    long double t = 1000, gamma = 0.999;
    //vector<long double> temperature = {t};
    int cost = 0;
    while (t > 0.001 && !a.empty()) {
        /*if (temperature.back()/t > 2) {
            temperature.push_back(t);
            cout << t << endl;
        }*/
        int ind = randint() % a.size();
        int will_cost = change(ind, a);
        if (will_cost >= cost || P(cost, will_cost, t)) {
                change(ind, a);
                cost = will_cost;
            }
            t *= gamma;
        }
        return a;
    }
```

CHT

```
#define ld long double
const ld E = 1e-8;

struct Line {
    int k, b;

    int value(int x) const {
        return k*x+b;
    }

    ld value(ld x) const {
        return k*x+b;
    }
};

ld inter(Line a, Line b) {
    return (ld)(b.b-a.b)/(a.k-b.k);
}

struct CHT { // max, different angles
    deque<pair<Line, ld>> a;

    void add_increasing(Line line) {
        while (a.size() > 1 && a.back().first.value(a.back().second)-E <
                                line.value(a.back().second)) {
            a.pop_back();
        }
        if (a.empty()) {
            a.emplace_back(line, -INF);
        } else {
            a.emplace_back(line, inter(a.back().first, line));
        }
    }

    void add_decreasing(Line line) {
        while (a.size() > 1 && a[1].first.value(a[1].second)-E < line.value(a[1].second)) {
            a.pop_front();
        }
        if (!a.empty()) {
            a[0].second = inter(a[0].first, line);
        }
        a.emplace_front(line, -INF);
    }

    int ans(int x) const {
        auto lambda = [](pair<Line, ld> x, ld y) {return x.second < y;};
        auto w = --lower_bound(a.begin(), a.end(), x, lambda);
        return w->first.value(x);
    }
};
```

Li_Chao

```
struct Line {
    int k, b;

    Line(): k(0), b(INF) {}
    Line(int k, int b): k(k), b(b) {}

    int value(int x) const {
        return k*x+b;
    }
};

struct Node {
    Line line;
    Node *l, *r;

    explicit Node(Line line): line(line), l(nullptr), r(nullptr) {}

    void make_sons() {
        if (l == nullptr) {
            l = new Node(Line());
        }
        if (r == nullptr) {
            r = new Node(Line());
        }
    }
};

struct Li_Chao { // min
    Node* root;
    int MIN, MAX;

    Li_Chao(int MIN, int MAX): MIN(MIN), MAX(MAX), root(new Node(Line())) {}

    static void add(int l, int r, Node* tree, Line line) {
        int m = (l+r)/2;
        if (line.value(m) < tree->line.value(m)) {
            swap(line, tree->line);
        }
        if (r-l == 1) {
            return;
        }
        tree->make_sons();
        if (line.k < tree->line.k) {
            add(m, r, tree->r, line);
        } else {
            add(l, m, tree->l, line);
        }
    }

    void add(Line line) const {
        add(MIN, MAX, root, line);
    }

    static int ans(int l, int r, Node* tree, int x) {
        int answer = tree->line.value(x);
        if (r-l == 1) {
            return answer;
        }
        tree->make_sons();
        int m = (l+r)/2;
        if (x < m) {
            answer = min(answer, ans(l, m, tree->l, x));
        } else {
            answer = min(answer, ans(m, r, tree->r, x));
        }
        return answer;
    }

    int ans(int x) const {
        return ans(MIN, MAX, root, x);
    }
};
```

Lileland_migration

```
vector<int> left_migration_less(vector<int> &a) {
    int n = a.size();
    vector<int> left(n, -1);
    vector<p> stack;
    for (int q = n-1; q > -1; q--) {
        while (!stack.empty() && stack.back().first > a[q]) {
            left[stack.back().second] = q;
            stack.pop_back();
        }
        stack.push_back({a[q], q});
    }
    return left;
}

vector<int> right_migration_less(vector<int> &a) {
    int n = a.size();
    vector<int> right(n, n);
    vector<p> stack;
    for (int q = 0; q < n; q++) {
        while (!stack.empty() && stack.back().first > a[q]) {
            right[stack.back().second] = q;
            stack.pop_back();
        }
        stack.push_back({a[q], q});
    }
    return right;
}
```

something_useful

```
unsigned seed = chrono::steady_clock::now().time_since_epoch().count();





#include<bits/extc++.h>
#define int long long
#define p pair<int, int>
#define endl '\n'
const int INF = 1000000001;

using namespace __gnu_pbds;
using namespace std;

typedef tree<int, null_type, less<>, rb_tree_tag, tree_order_statistics_node_update> ordered_set;
```

```
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,popcnt")
// avx2 -> avx/sse/sse2/sse3/sse4
// popcnt -> lzcnt/bmi/bmi2/...





set(CMAKE_CXX_FLAGS -Wl,--stack=2000000179)
set(CMAKE_CXX_FLAGS -fsplit-stack)
```

Aho_Corasick

```
int C = 26;
vector<vector<int>> d, term;
vector<int> suf_link, num_term, term_link;

void make_trie(vector<string> &a) {
    int n = a.size();
    d = {vector<int>(C, 1), vector<int>(C, -1)}, term = {{}, {}};
    for (int q = 0; q < n; q++) {
        string s = a[q];
        int vertex = 1;
        for (char q2 : s) {
            if (d[vertex][q2-'a'] == -1) {
                d[vertex][q2-'a'] = d.size();
                d.push_back(vector<int>(C, -1));
                term.emplace_back();
            }
            vertex = d[vertex][q2-'a'];
        }
        term[vertex].push_back(q);
    }
}

void Aho_Corasick() {
    int n = d.size();
    suf_link.assign(n, -1), num_term.assign(n, -1), term_link.assign(n, -1);
    suf_link[0] = suf_link[1] = 0, num_term[0] = 0;
    queue<int> a;
    a.push(1);
    while (!a.empty()) {
        int x = a.front();
        a.pop();
        for (int q = 0; q < C; q++) {
            if (d[x][q] != -1) {
                int q1 = suf_link[x];
                while (d[q1][q] == -1) {
                    q1 = suf_link[q1];
                }
                suf_link[d[x][q]] = d[q1][q];
                a.push(d[x][q]);
            }
        }
        if (!term[suf_link[x]].empty()) {
            term_link[x] = suf_link[x];
        } else {
            term_link[x] = term_link[suf_link[x]];
        }
        num_term[x] = num_term[suf_link[x]]+term[x].size();
    }
}
```

Manacher

```
vector<int> Manacher(vector<int> &a1) {
    int n = a1.size();
    vector<int> a = {a1[0]};
    for (int q = 1; q < n; q++) {
        a.push_back(INF);
        a.push_back(a1[q]);
    }
    n = a.size();
    vector<int> man = {1};
    int ind = 0;
    for (int q = 1; q < n; q++) {
        int k = q;
        if (ind+man[ind] > q) {
            k = min(man[ind]+ind, q+man[2*ind-q]);
        }
        while (k < n && 2*q-k > -1 && a[k] == a[2*q-k]) {
            k++;
        }
        man.push_back(k-q);
        if (k > man[ind]+ind) {
            ind = q;
        }
    }
    for (int q = 0; q < n; q++) {
        if (a[q] == INF) {
            man[q] -= man[q] % 2;
        } else {
            man[q] -= 1-man[q] % 2;
        }
    }
    return man;
}
```

Prefix_function

```
vector<int> pref_func(string &a) {
    vector<int> pref = {0};
    for (int q1 = 1; q1 < a.size(); q1++) {
        char q = a[q1];
        int now = pref.back();
        while (now > 0 && q != a[now]) {
            now = pref[now-1];
        }
        if (q == a[now]) {
            pref.push_back(now+1);
        } else {
            pref.push_back(0);
        }
    }
    return pref;
}
```

Suffix_array

```
vector<int> suf_mas(string &s) {
    s += '#';
    int n = s.size();
    vector<int> suf(n);
    iota(suf.begin(), suf.end(), 0);
    sort(suf.begin(), suf.end(), [&s](int x, int y) {return s[x] < s[y];});
    vector<int> cls(n, 0);
```

```
        for (int q = 1; q < n; q++) {
            cls[suf[q]] = cls[suf[q-1]]+(s[suf[q-1]] < s[suf[q]]);
        }
        int deg = 1;
        while (cls[suf.back()] < n-1) {
            vector<int> nums(cls[suf.back()]+1, 0);
            for (int q : suf) {
                nums[cls[q-deg+(q < deg)*n]]++;
            }
            vector<int> ind(cls[suf.back()]+1, 0);
            for (int q = 1; q <= cls[suf.back()]; q++) {
                ind[q] = ind[q-1]+nums[q-1];
            }
            vector<int> will_suf(n);
            for (int q : suf) {
                will_suf[ind[cls[q-deg+(q < deg)*n]]++] = q-deg+(q < deg)*n;
            }
            vector<int> will_cls(n, 0);
            for (int q = 1; q < n; q++) {
                bool change = (cls[will_suf[q-1]] != cls[will_suf[q]] ||
                               cls[will_suf[q-1]+deg-(will_suf[q-1]+deg >= n)*n] !=
                               cls[will_suf[q]+deg-(will_suf[q]+deg >= n)*n]);
                will_cls[will_suf[q]] = will_cls[will_suf[q-1]]+change;
            }
            suf = will_suf, cls = will_cls, deg *= 2;
        }
        s.pop_back();
        return suf;
    }

    vector<int> LCP(string &s, vector<int> &sufmas) {
        s += '#';
        int n = s.size();
        vector<int> where(n);
        for (int q = 0; q < n; q++) {
            where[sufmas[q]] = q;
        }
        vector<int> lcp(n-1, 0);
        for (int q = 0; q < n-1; q++) {
            if (where[q] == n-1) {
                continue;
            }
            int is = (q == 0 || where[q-1] == n-1 ? 0 : max(0LL, lcp[where[q-1]]-1));
            for (int q1 = is;; q1++) {
                if (s[q+q1] != s[sufmas[where[q]+1]+q1]) {
                    lcp[where[q]] = q1;
                    break;
                }
            }
        }
        s.pop_back();
        return lcp;
    }

    pair<p, int> find(string &s, string &t, vector<int> &sufmas) {
        s += '#';
        int n = sufmas.size(), m = t.size();
        int l = 0, r = n, ind = 0;
        for (; ind < m; ind++) {
            auto lambda_l = [&s, ind](int x, char y) {return s[x+ind] < y;};
            int l1 = lower_bound(sufmas.begin()+l, sufmas.begin()+r, t[ind], lambda_l)-sufmas.begin();
            auto lambda_r = [&s, ind](int x, char y) {return s[x+ind] <= y;};
            int r1 = lower_bound(sufmas.begin()+l, sufmas.begin()+r, t[ind], lambda_r)-sufmas.begin();
            if (l1 == r1) {
```

```
            break;
        }
        l = l1, r = r1;
    }
    s.pop_back();
    return {{l, r}, ind};
}
```

Suffix_automaton

```
int C = 26, L;

struct Node {
    vector<int> a;
    int suf, len, right, parent;
    char symbol;

    Node(int parent1 = -1, char symbol1 = ’#’, int len1 = 0, vector<int> a1 = {}) {
        a = (a1.empty() ? vector<int>(C, -1) : a1);
        parent = parent1, symbol = symbol1, len = len1+1, suf = -1, right = 0;
    }
};

vector<Node> trie;
vector<int> last;

int make_moves(int &vertex, char q) {
    if (trie[vertex].a[q-’a’] != -1) {
        return trie[vertex].a[q-’a’];
    }
    int b = trie.size();
    trie.push_back(Node(vertex, q, trie[vertex].len));
    while (vertex != -1 && trie[vertex].a[q-’a’] == -1) {
        trie[vertex].a[q-’a’] = b;
        vertex = trie[vertex].suf;
    }
    trie[b].suf = (vertex == -1 ? 0 : trie[vertex].a[q-’a’]);
    return b;
}

int add(int vertex, char q) {
    int b = make_moves(vertex, q);
    if (vertex == -1 || trie[trie[vertex].a[q-’a’]].parent == vertex) {
        return b;
    }
    int c = trie[vertex].a[q-’a’], d = trie.size();
    trie.push_back(Node(vertex, q, trie[vertex].len, trie[c].a));
    while (vertex != -1 && trie[vertex].a[q-’a’] == c) {
        trie[vertex].a[q-’a’] = d;
        vertex = trie[vertex].suf;
    }
    trie[d].suf = trie[c].suf, trie[c].suf = trie[b].suf = d;
    return (b == c ? d : b);
}

vector<bool> was;

void build_other(int vertex) {
    was[vertex] = true;
    for (int q = 0; q < C; q++) {
        if (trie[vertex].a[q] == -1) {
            continue;
        }
        if (!was[trie[vertex].a[q]]) {
            build_other(trie[vertex].a[q]);
        }
        trie[vertex].right += trie[trie[vertex].a[q]].right;
    }
}

void build_automation(vector<string> &a) {
    trie = {Node()};
    for (string &s : a) {
        int vertex = 0;
        for (char &q : s) {
            vertex = add(vertex, q);
        }
        last.push_back(vertex);
    }
    L = trie.size();
    for (int q = 0; q < (int)a.size(); q++) {
        int vertex = last[q];
        while (vertex != -1) {
            trie[vertex].right++;
            vertex = trie[vertex].suf;
        }
    }
    was.assign(L, false);
    build_other(0);
}
```

Z_function

```
vector<int> z_func(string &a) {
    vector<int> z = {0};
    int ind = 0;
    for (int q1 = 1; q1 < a.size(); q1++) {
        if (ind+z[ind] >= q1 && q1+z[q1-ind] < ind+z[ind]) {
            z.push_back(z[q1-ind]);
        } else {
            int q2 = max(q1, ind+z[ind]);
            while (q2 < a.size() && a[q2] == a[q2-q1]) {
                q2++;
            }
            z.push_back(q2-q1);
            ind = q1;
        }
    }
    return z;
}
```

Fenwick

```
struct Fen {
    vector<int> fen;
    int n;

    Fen(int n1) {
        n = n1+1;
        fen.assign(n, 0);
    }

    void plus(int q, int x) {
        for (++q; q < n; q += (q & -q)) {
            fen[q] += x;
        }
    }

    int sum(int q) {
        int res = 0;
        for (; q > 0; q -= (q & -q)) {
            res += fen[q];
        }
        return res;
    }

    int sum(int l, int r) {
        return sum(r)-sum(l);
    }
};
```

DO_bottom_up

```
struct DO {
    vector<int> do_arr, mins;
    vector<bool> degs;
    int len;

    DO(vector<int> a) {
        len = 1;
        while (len < a.size()) {
            len <<= 1;
        }
        do_arr.assign(len << 1, 0), mins.assign(len << 1, INF);
        degs.assign(len << 1, false), degs[len] = true;
        for (int q = len; q < a.size()+len; q++) {
            do_arr[q] = mins[q] = a[q-len];
        }
        for (int q = len-1; q > 0; q--) {
            do_arr[q] = do_arr[q << 1]+do_arr[(q << 1)+1];
            mins[q] = min(mins[q << 1], mins[(q << 1)+1]);
            degs[q] = degs[q << 1];
        }
    }

    void change(int q, int x) {
        do_arr[q+len] = x, mins[q+len] = x;
        q = ((q+len) >> 1);
        while (q > 0) {
            do_arr[q] = do_arr[q << 1]+do_arr[(q << 1)+1];
            mins[q] = min(mins[q << 1], mins[(q << 1)+1]);
            q >>= 1;
        }
    }

    int ans(int l, int r) {
        if (l >= r) {
            return 0;
        }
        l += len, r += len-1;
        int sum1 = 0;
        while (l < r) {
            sum1 += (l & 1)*do_arr[l];
            l = ((l+1) >> 1);
            sum1 += ((r & 1) ^ 1)*do_arr[r];
            r = ((r-1) >> 1);
        }
        return sum1+(l == r)*do_arr[l];
    }

    int left_less(int q, int x) {
        q += len;
        while (!degs[q] && mins[q] >= x) {
            q = ((q-1) >> 1);
        }
        if (mins[q] >= x) {
            return -1;
        }
        while (q < len) {
            q = (q << 1)+(mins[(q << 1)+1] < x);
        }
        return q-len;
    }
};
```

Implicit_DO

```
struct Node {
    int left, right;
    Node *l, *r;
    int size;

    Node(int left1, int right1):
        left(left1), right(right1), l(nullptr), r(nullptr), size(0) {}
};

void make_sons(Node* tree) {
    int m = ((tree->left+tree->right) >> 1);
    if (tree->l == nullptr) {
        tree->l = new Node(tree->left, m);
    }
    if (tree->r == nullptr) {
        tree->r = new Node(m, tree->right);
    }
}

void add(Node* tree, int x) {
    int l = tree->left, r = tree->right;
    tree->size++;
    if (r-l == 1) {
        return;
    }
    make_sons(tree);
    int m = ((l+r) >> 1);
    if (x < m) {
        add(tree->l, x);
    } else {
        add(tree->r, x);
    }
}

void del(Node* tree, int x) {
    int l = tree->left, r = tree->right;
    tree->size--;
    if (r-l == 1) {
        return;
    }
    make_sons(tree);
    int m = ((l+r) >> 1);
    if (x < m) {
        del(tree->l, x);
    } else {
        del(tree->r, x);
    }
}
```

```

int num_x(Node* tree, int x) {
    int l = tree->left, r = tree->right;
    if (r-l == 1) {
        return tree->size*(x == l);
    }
    make_sons(tree);
    int m = ((l+r) >> 1);
    if (x < m) {
        return num_x(tree->l, x);
    }
    return num_x(tree->r, x)+tree->l->size;
}
```

pointers

```
struct Node {
    int x;
    Node *l, *r;

    Node(int x1 = 0): x(x1), l(nullptr), r(nullptr) {}
};

void update(Node* tree) {
    tree->x = (tree->l != nullptr ? tree->l->x : 0)+(tree->r != nullptr ? tree->r->x : 0);
}

Node* build(int l, int r) {
    Node* now = new Node();
    if (r-l == 1) {
        return now;
    }
    int m = (l+r)/2;
    now->l = build(l, m), now->r = build(m, r);
    update(now);
    return now;
}

Node* change(Node* tree, int l, int r, int q, int x) {
    if (r-l == 1) {
        return new Node(x);
    }
    Node* now = new Node();
    int m = (l+r)/2;
    if (q < m) {
        now->l = change(tree->l, l, m, q, x), now->r = tree->r;
    } else {
        now->l = tree->l, now->r = change(tree->r, m, r, q, x);
    }
    update(now);
    return now;
}

int ans(Node* tree, int l, int r, int l1, int r1) {
    if (l1 >= r || l >= r1) {
        return 0;
    }
    if (l1 <= l && r <= r1) {
        return tree->x;
    }
    int m = (l+r)/2;
    return ans(tree->l, l, m, l1, r1)+ans(tree->r, m, r, l1, r1);
}
```

DD

```
mt19937 randint(179);

struct Node {
    int x, y;
    Node *l, *r;
    int size;

    Node(int x1): x(x1), y(randint()), l(nullptr), r(nullptr), size(1) {}
};

void update(Node* tree) {
    if (tree == nullptr) {
        return;
    }
    tree->size = 1;
    if (tree->l != nullptr) {
        tree->size += tree->l->size;
    }
    if (tree->r != nullptr) {
        tree->size += tree->r->size;
    }
}

Node* merge(Node* tree1, Node* tree2) {
    if (tree1 == nullptr) {
        return tree2;
    }
    if (tree2 == nullptr) {
        return tree1;
    }
    if (tree1->y < tree2->y) {
        tree1->r = merge(tree1->r, tree2);
        update(tree1);
        return tree1;
    }
    tree2->l = merge(tree1, tree2->l);
    update(tree2);
    return tree2;
}

pair<Node*, Node*> split(Node* tree, int x) {
    if (tree == nullptr) {
        return {nullptr, nullptr};
    }
    if (tree->x <= x) {
        pair<Node*, Node*> trees = split(tree->r, x);
        tree->r = trees.first;
        update(tree);
        return {tree, trees.second};
    }
    pair<Node*, Node*> trees = split(tree->l, x);
    tree->l = trees.second;
    update(tree);
    return {trees.first, tree};
}

pair<Node*, Node*> split_num(Node* tree, int k) {
    if (tree == nullptr) {
        return {nullptr, nullptr};
    }
    int t = (tree->l != nullptr ? tree->l->size : 0);
    if (t < k) {
        pair<Node*, Node*> trees = split_num(tree->r, k-t-1);
        tree->r = trees.first;
        update(tree);
        return {tree, trees.second};
    }
    pair<Node*, Node*> trees = split_num(tree->l, k);
    tree->l = trees.second;
    update(tree);
    return {trees.first, tree};
}

Node* add(Node* tree, int x) {
    pair<Node*, Node*> trees = split(tree, x);
    Node* now = new Node(x);
    return merge(merge(trees.first, now), trees.second);
}

Node* del(Node* tree, int x) {
    pair<Node*, Node*> trees = split(tree, x);
    int k = (trees.first != nullptr ? trees.first->size-1 : 0);
    pair<Node*, Node*> trees1 = split_num(trees.first, k);
    return merge(trees1.first, trees.second);
}

int num_x(Node* tree, int x) {
    int ans = 0;
    while (tree != nullptr) {
        if (tree->x <= x) {
            ans += (tree->l != nullptr ? tree->l->size : 0)+1;
            tree = tree->r;
        } else {
            tree = tree->l;
        }
    }
    return ans;
}
```

Implicit_DD

```
mt19937 randint(179);

struct Node {
    int x, y;
    Node *l, *r, *parent;
    int size, sum;

    Node(int x1): x(x1), y(randint()), l(nullptr), r(nullptr), parent(nullptr), size(1), sum(x) {}
};

void update(Node* tree) {
    if (tree == nullptr) {
        return;
    }
    tree->size = 1, tree->sum = tree->x;
    if (tree->l != nullptr) {
        tree->size += tree->l->size, tree->sum += tree->l->sum;
    }
    if (tree->r != nullptr) {
        tree->size += tree->r->size, tree->sum += tree->r->sum;
    }
}

void change_parent(Node* tree, Node* parent) {
    if (tree == nullptr) {
            return;
        }
        tree->parent = parent;
    }

    void change_left(Node* tree, Node* left) {
        if (tree == nullptr) {
            return;
        }
        tree->l = left;
        change_parent(left, tree);
        update(tree);
    }

    void change_right(Node* tree, Node* right) {
        if (tree == nullptr) {
            return;
        }
        tree->r = right;
        change_parent(right, tree);
        update(tree);
    }

    Node* merge(Node* tree1, Node* tree2) {
        if (tree1 == nullptr) {
            return tree2;
        }
        if (tree2 == nullptr) {
            return tree1;
        }
        if (tree1->y < tree2->y) {
            change_parent(tree1->r, nullptr);
            change_right(tree1, merge(tree1->r, tree2));
            return tree1;
        }
        change_parent(tree2->l, nullptr);
        change_left(tree2, merge(tree1, tree2->l));
        return tree2;
    }

    pair<Node*, Node*> split(Node* tree, int k) {
        if (tree == nullptr) {
            return {nullptr, nullptr};
        }
        int t = (tree->l != nullptr ? tree->l->size : 0);
        if (k <= t) {
            change_parent(tree->l, nullptr);
            pair<Node*, Node*> trees = split(tree->l, k);
            change_left(tree, trees.second);
            return {trees.first, tree};
        }
        change_parent(tree->r, nullptr);
        pair<Node*, Node*> trees = split(tree->r, k-t-1);
        change_right(tree, trees.first);
        return {tree, trees.second};
    }

    Node* add(Node* tree, int k, Node* vertex) {
        pair<Node*, Node*> trees = split(tree, k);
        return merge(merge(trees.first, vertex), trees.second);
    }

    Node* del(Node* tree, int k) {
        pair<Node*, Node*> trees1 = split(tree, k);
        pair<Node*, Node*> trees2 = split(trees1.second, 1);
        return merge(trees1.first, trees2.second);
    }

    Node* root(Node* tree) {
        if (tree == nullptr) {
            return nullptr;
        }
        while (tree->parent != nullptr) {
            tree = tree->parent;
        }
        return tree;
    }

    int find_pos(Node* tree) {
        if (tree == nullptr) {
            return 0;
        }
        int ans = (tree->l != nullptr ? tree->l->size : 0);
        while (tree->parent != nullptr) {
            if (tree->parent->l != tree) {
                ans += (tree->parent->l != nullptr ? tree->parent->l->size : 0)+1;
            }
            tree = tree->parent;
        }
        return ans;
    }

    Node* find_element(Node* tree, int k) {
        if (tree == nullptr) {
            return nullptr;
        }
        int t = (tree->l != nullptr ? tree->l->size : 0);
        if (k == t) {
            return tree;
        }
        if (k < t) {
            return find_element(tree->l, k);
        }
        return find_element(tree->r, k-t-1);
    }
```

Persistent_DD

```
    mt19937 randint(179);

    struct Node {
        int x, size;
        Node *l, *r;

        Node(int x1): x(x1), size(1), l(nullptr), r(nullptr) {}
    };

    void update(Node* tree) {
        if (tree == nullptr) {
            return;
        }
        tree->size = 1;
        if (tree->l != nullptr) {
            tree->size += tree->l->size;
        }
        if (tree->r != nullptr) {
            tree->size += tree->r->size;
        }
    }

    Node* copy(Node* tree) {
        if (tree == nullptr) {
            return nullptr;
        }
        Node* now = new Node(tree->x);
        now->l = tree->l, now->r = tree->r;
        update(now);
        return now;
    }

    Node* merge(Node* tree1, Node* tree2) {
        if (tree1 == nullptr) {
            return tree2;
        }
        if (tree2 == nullptr) {
            return tree1;
        }
        if (randint() % (tree1->size+tree2->size) < tree1->size) {
            Node* now = copy(tree1);
            now->r = merge(tree1->r, tree2);
            update(now);
            return now;
        }
        Node* now = copy(tree2);
        now->l = merge(tree1, tree2->l);
        update(now);
        return now;
    }

    pair<Node*, Node*> split(Node* tree, int k) {
        if (tree == nullptr) {
            return {nullptr, nullptr};
        }
        int left = (tree->l == nullptr ? 0 : tree->l->size);
        Node* now = copy(tree);
        if (k <= left) {
            pair<Node*, Node*> trees = split(tree->l, k);
            now->l = nullptr;
            update(now);
            trees.second = merge(trees.second, now);
            return trees;
        }
        pair<Node*, Node*> trees = split(tree->r, k-left-1);
        now->r = nullptr;
        update(now);
        trees.first = merge(now, trees.first);
        return trees;
    }

    pair<Node*, bool> change(Node* tree, int pos) {
        auto trees1 = split(tree, pos+1);
        auto trees2 = split(trees1.first, pos);
        Node* will = copy(trees2.second);
        bool flag = (will->x == 1);
        will->x = 1-will->x;
        return {merge(merge(trees2.first, will), trees1.second), flag};
    }
```

Сумма Минковского

```
    Polygon operator+(Polygon a, Polygon b) {
        a.arr.push_back(a.arr[0]);
        b.arr.push_back(b.arr[0]);
        a.arr.push_back(a.arr[1]);
        b.arr.push_back(b.arr[1]);
        int i = 0, j = 0;
        Polygon ans;
        while (i < a.n || j < b.n) {
            ans.arr.push_back(a[i] + b[j]);
            if (((Vect(a[i], a[i + 1])) ^ Vect(b[j], b[j + 1])) < 0) {
                j++;
                continue;
            }
            if (((Vect(a[i], a[i + 1])) ^ Vect(b[j], b[j + 1])) > 0) {
                i++;
                continue;
            }
            i++;
            j++;
        }
        ans.n = ans.arr.size();
        return ans;
    }
```

ICPC_algorithms

```
    ./graph/Bridges.cpp
    ./graph/CSS.cpp
    ./graph/flow/Dinitz.cpp
    ./graph/flow/Min_cost.cpp
    ./graph/Kun.cpp
    ./graph/tree/Centroid.cpp
    ./graph/tree/HLD.cpp
    ./graph/tree/LCA/LCA_linear_memory.cpp
    ./math/Berlekamp.cpp
    ./math/delivers/Pollard.cpp
    ./math/FFT/FFT_divide.cpp
    ./math/FFT/FFT_modulo.cpp
    ./math/functions/C_k_n_modulo_p.cpp
    ./math/functions/N_th_root.cpp
    ./math/geometry/double.cpp
    ./math/geometry/polygons.cpp
    ./math/matrix/Binary_Gauss.cpp
    ./math/matrix/SLAE_solve_module.cpp
    ./math/Or_And_convolution.cpp
    ./math/Smiths_theory.cpp
    ./other/Annealing_simulation.cpp
    ./other/dp/CHT.cpp
    ./other/dp/Li_Chao.cpp
    ./other/Lileland_migration.cpp
    ./other/something_useful.cpp
    ./string/Aho_Corasick.cpp
    ./string/Manacher.cpp
    ./string/Prefix_function.cpp
    ./string/Suffix_array.cpp
    ./string/Suffix_automaton.cpp
    ./string/Z_function.cpp
    ./struct/Fenwick.cpp
    ./struct/Segment_Tree/DO_bottom_up.cpp
    ./struct/Segment_Tree/Implicit_DO.cpp
    ./struct/Segment_Tree/persistent/pointers.cpp
    ./struct/Treap/DD.cpp
    ./struct/Treap/Implicit_DD.cpp
```